Programming Techniques for Supercomputers

Exam informtaion

Erlangen National High Performance Computing Center

Department of Computer Science

FAU Erlangen-Nürnberg

Sommersemester 2025

- Time:
  - *5 ECTS (lecture only): 60 minutes*
  - 7.5 ECTS (lecture + tutorials): 90 minutes
  - Same questions for both + additional questions (covering tutorials) for 7.5

- Permitted aids
  - Pen & ruler
  - Simple (non-programmable) calculator
  - YOUR BRAIN
- All code will be given in C

- General motivation of exam questions:
  Understand basic architectural concepts, relevant code transformations and the interaction of both resulting in actual performance (making sense of performance)

- Do not try to learn numbers and terms without understanding them

- Be familiar with basic concepts

```
#pragma omp parallel for
```

Question 1.a.: Parallelize the following subroutine using OpenMP

```
double dmvm(int n, int m, double *lhs, double *rhs, double *mat, double
sum){

double s=0.;

for(r=0; r<n; ++r) {

    offset=m*r;

    for(c=0; c<m; ++c)

      lhs[r] += mat[c + offset]*rhs[c];

      s = s + lhs[r]*rhs[r];
}

return sum+s;}
```

Perfect answer

```
double dmvm(int n, int m, double *lhs, double *rhs, double *mat,
            double sum){

double s=0.;

#pragma omp parallel for private(offset,c) reduction(+:s)

for(r=0; r<n; ++r) {

    offset=m*r;

    for(c=0; c<m; ++c)

      lhs[r] += mat[c + offset]*rhs[c];

    s = s + lhs[r]*rhs[r];
}

return sum+s;

}
```

Also valid / good answer

```
double dmvm(int n, int m, double *lhs, double *rhs, double *mat,
            double sum){

double s=0.;

#pragma omp parallel for private(offset,c) reduction*

for(r=0; r<n; ++r) {

    offset=m*r;

    for(c=0; c<m; ++c)

      lhs[r] += mat[c + offset]*rhs[c];

      s = s + lhs[r]*rhs[r];
}

return sum+s;}

*: I do not know exact syntax but I need a reduction operation on s
```

Learn basic concepts!

- Question 1.b.: What will be the maximum performance of this code for n=m=20,000 on a multicore processor chip with
  - b_S=48 GB/s bandwidth and an
  - L3 cache of 20 MiB ?

→ RLM / correct computation of I or B_C

- Question 1.c.: Your OpenMP parallel subroutine is called from a code which runs on a 2-socket node of ccNUMA architecture. Assume large n and m: Does performance always perfectly scale from 1 to 2 sockets?

→ ccNUMA / First touch

Learn basic concepts

- Question 1.d.: Optimize single core execution for a minimum L2 code balance, assuming an L1 cache of 32 KiB.

→ Blocking for RHS

Learn basic concepts

- Question 2.a.: Briefly describe the first touch concept in ccNUMA architectures

- Question 2.b.: What implication does this concept have on shared memory parallel *programming* for ccNUMA nodes if we aim for optimal performance?

- Question 3.a.: Briefly describe the concept of superscalarity!

- Question 3.b.: Name one performance metric that quantifies the level of superscalarity in a running code!

- Question 3.c.: Given an architecture (specs) – can the following code fully exploit superscalarity?

Learn basic concepts

- For a given code and architectural description
  - build the *refined/extended* roofline model or calcultae P_{max}
  - Draw the *basic* roofline graph for this machine: clock speed, FMA, SIMD, cores, memory bandwidth
  - How does it change if SIMD is disabled....?

- Performance of a simple benchmark as a function of loop length is given: Determine how many cache levels & of which size?

- If you run that OpenMP parallel code, what do you need to consider to get reliable / useful performance numbers
  - Exclusive / Clock speed / Affinity-pinning / variations / reasonable-sensibility

A selection of very important topics

- General:
  - Performance and work metrics
  - Impact factor & best practices for performance measurements

- Single Core:
  - Basic resource bottlenecks of code execution
  - Pipelining
  - Superscalarity & OOO
  - SIMD
  - Performance composition: P_{core}
  - **P_max calculation**
  - Do not learn processors specs! You should only know typical numbers!

A selection of very important topics

- Memory hierarchy – data access:
  - Effective BW model – Hockney’s law
  - Prefetching & Spatial/temporal locality & memory layout
  - Basics of cache mapping (direct, m-may set-assoc.)
  - Cache thrashing – when may it happen

- Data access optimization:
  - Performance / Balance if data comes from L1, L2 or L3 cache
  - cache sizes and LD/ST throughput
  - **Dense MVM traffic analysis + Blocking**
  - Algorithm classification

A selection of very important topics

- Shared Memory Parallel:
  - P_{Chip}
  - Shared memory architectures: UMA vs ccNUMA
  - Topology / Pinning
  - **Not relevant for exam: Cache coherence & False Sharing**

A selection of very important topics

- OpenMP Basics:
  - Everybody should be able to parallelize a simple kernel with OpenMP „correctly“ and achieve good performance on modern architectures
  - Basic ideas, keywords, concepts
  - **Not relevant for exam: ordered, locks, threadprivate**

- GPU:
  - Basic difference CPU / GPU – CUDA vs OpenMP
  - Ppeak & #threads to saturate main memory bandwidth
  - Tune the execution configuration (CUDA thread block size and number of thread blocks) for a given application
  - Memory coalescing

A selection of very important topics

- Roofline - **Important**
  - Assumptions!!!
  - Naïve and refined/extended Roofline Model (RLM) : Immensely important!
- Stencils – **Important**
  - RL analysis, Layer Conditions (!), LCs & shared caches
  - Overhead with middle loop parallelization

A selection of very important topics

- SpMV
  - Basic performance problems + performance modelling!
  - Basic data layout considerations – CPU vs. GPU

- Amdahl/Gustafson:
  - Amdahl’s & Gustafson’s Laws, incl. communication overhead
  - How to correctly „measure“ serial parts
  - (Energy model – tutorial)
  - (slow computing – tutorial)

A selection of very important topics

- Advanced OpenMP
  - ccNUMA concept of data placement
  - Efficient ccNUMA programming
  - Page migration
  - NO false sharing

Additional material from the tutorials

- Power dissipation issues
  - f^3 law for dynamic power
  - Multicore energy model (baseline power, dynamic power)
  - Relevance of performance saturation for energy consumption

- Slow computing
  - Machine with slower CPUs scales better
  - Slow code scales better
  - Why?